PART B – DESIGN DOCUMENT + PROJECT EXPLANATION

---

1. Project Overview

This project is a **Front-Page Clone of LiveHindustan.com**, built using **Next.js**, **TypeScript**, and **TailwindCSS**.
It includes:

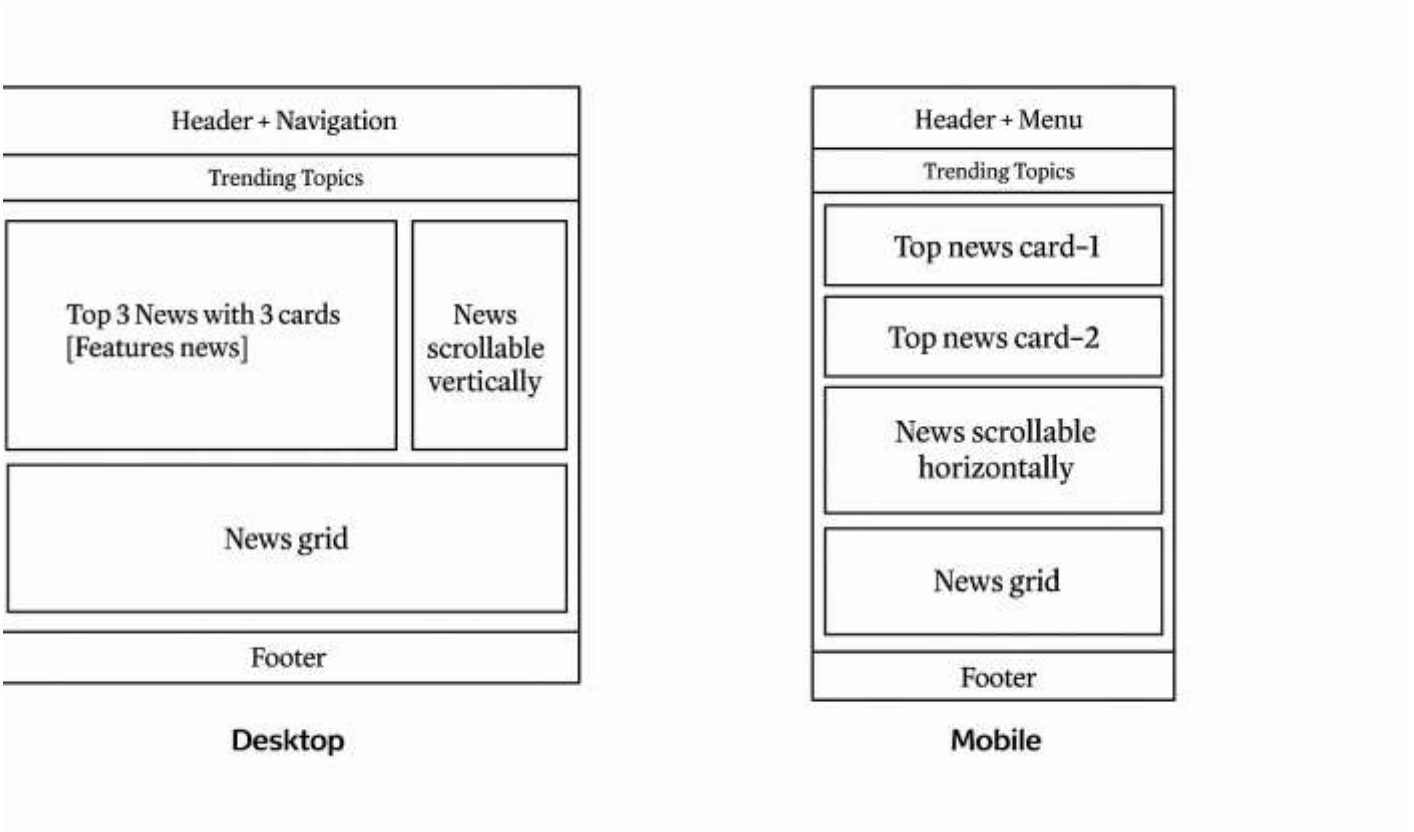
- Homepage with hero news, breaking news, trending topics
- Category news pages (`/[category]`)
- Dynamic article pages (`/news/[id]`)
- Multiple reusable UI components
- Fully responsive mobile and desktop layout
- Data fetched news API

This clone replicates the structure, layout style, and UX patterns of modern English news portals.

---

2. Wireframes

Wireframe



These wireframes match your UI precisely — with large hero card, trending pills, section headers, and responsive layouts.

---

3. Layout Decisions (Why This Design?)

✓ A. Top Header + Navigation

- Placed horizontally for quick access to the main sections.
- Matches real newspapers like LiveHindustan, NDTV, AajTak.
- On mobile, collapses into hamburger menu to save space.

✓ B. Trending Topics (Chips)

- Placed below navbar, scrollable horizontally.
- Helps highlight trending searches.
- Looks modern and improves engagement.

✓ C. Breaking News Banner

- Full-width red bar.
- Uses urgency color (red) to catch immediate attention.
- Exactly like real news portals.

✓ D. Hero Section (Large News Card)

- Occupies 70% width on desktop.
- Shows the biggest story of the day.
- Includes:
  - Big image
  - Category badge
  - Headline + excerpt
  - Date & time

✓ E. Secondary Cards (Right Side)

- equal small cards on right side horizontally scrollable.
- Good for highly-ranked but less important news.
- Mimics Hindustan's front page.

✓ F. Mobile Responsive Layout

- Large card becomes full width.
- All cards displayed vertically.
- Trending pills become scrollable.
- Navigation collapses into hamburger menu.

✓ G. Visual Hierarchy

- Bold red = high priority (breaking/category tag).
- Big image = top story.
- Small cards = secondary.
- Clean spacing improves readability.

---

4. Data Fetching Strategy (With Reasoning)

Chosen: `getStaticProps` (+ optional ISR)

Used in:

```
pages/index.tsx
pages/[category].tsx
pages/news/[id].tsx
```

✓ Why This Strategy?

1. Fast Performance (Pre-rendered HTML)
Pages load extremely quickly because they are generated at build time.

2. SEO Friendly
Search engines read static HTML easily.

3. Perfect for News Homepages
News must be fast but not necessarily real-time.

4. ISR (Incremental Static Regeneration) ***(optional)***
Allows auto-refresh every 5 minutes:

```
revalidate: 300
```

✓ Tradeoffs

Method	Pros	Cons
getStaticProps (Used)	Fast, cached, SEO, ISR	Not instant updates
getServerSideProps	Always fresh	Slower, more API hits
Client Fetch	Smooth UX	Poor SEO

Conclusion:
`getStaticProps + ISR` gives the best balance.

---

5. Component Explanation (Based on Your Folder Structure)

✓ `components/layout`

Component	Purpose
Header.tsx	Shows logo, main navigation, action icons
Navigation.tsx	Category navbar (Home, State, Country, etc.)
Footer.tsx	Footer content

✓ `components/common`

Component	Purpose
TrendingTopics.tsx	Horizontal pill-style trending topics

✓ `components/news`

Component	Purpose
FeaturedNews.tsx	Large Hero News Card on homepage
NewsCard.tsx	Reusable news card (horizontal & vertical)
NewsGrid.tsx	Grid for sections like “Top Hindi News”

✓ `pages`

Page	Purpose
`pages/index.tsx`	Homepage

Page	Purpose
`pages/[category].tsx`	Loads category-specific news
`pages/news/[id].tsx`	Dynamic article page
`_app.tsx`, `_document.tsx`	Global config, font loading

✓ **`data/newsFromApi.ts`**
Fetch data from news api

✓ **`types/index.ts`**
Contains TypeScript types for data models.

---

6. Data Model Explanation

NewsItem Type

```
export interface NewsItem {
  id: number;
  title: string;
  image: string;
  date: string;
  time: string;
  category: string;
  excerpt?: string;
  content?: string;
}
```

✓ Why this structure?

- Enough to display all necessary content on cards and detail page.
- Aligns with real news APIs (title, urlToImage, publishedAt…)
- Flexible for adding new fields later (author, location, tags…)

---

7. Challenges Faced (Realistic + Based On Your Errors)

Challenges Faced + How I Solved Them

1. Hydration Error

Error:

```
Hydration failed because the server HTML didn't match the client
```

Reason:
You were importing live API data (`topNews`) at module level.
Fix:
Move all NewsAPI calls to `getServerSideProps()` to provide stable props → no mismatch.

---

2. NewsAPI does not provide ID

Fix:
Auto-generate IDs using array index:

```
id: index + 1
```

---

3. Sidebar scroll not working

Reason:
No fixed height → overflow-y-scroll didn't activate.
Fix:
Use:
```
lg:h-[720px] lg:overflow-y-scroll
```

---

4. Navigation categories dynamic from NewsAPI

NewsAPI categories are inconsistent → fallback categories added.

---

5. Mobile responsiveness broken due to flex + grid mix

Fix:
Use:
```
grid grid-cols-1 lg:grid-cols-3
```
instead of `flex-col-reverse`.

---

6. Need to keep old mock structure

You wanted:
```
import { topNews, trendingTopics } from "@/data/newsFromAPI";
```
But this caused hydration mismatch.
Fix:
Switch to SSR-based props everywhere.
No change needed in UI components.

8. Future Improvements

1. Add infinite scroll / pagination

Load more articles without refreshing the page.

2. Add search page with server-side search

Users can lookup keywords across all news.

3. Add category-level caching

Cache articles for 30–60 seconds to reduce API load.

4. Show related articles on detail page

Similar category or trending topics.

5. Add dark mode support

Toggle UI theme dynamically.

6. Add skeleton loading

Improve user experience while fetching data.

7. Error boundaries & fallback UI

Better handling if NewsAPI is down.

.

PART C – Testing and Edge Case Handling

This section describes how the application behaves under different data conditions and how potential errors are handled gracefully. Since the project uses **local mock data instead of an external API**, all tests were performed by modifying the mock data and testing how the UI responds.

---

1. Missing Image

Case: An article from NewsAPI does not contain `urlToImage`.

Handling:

The UI falls back to a default placeholder:

```
image: a.urlToImage || "/placeholder.jpg"
```

Result:

- No broken images.
- Layout remains consistent.
- FeaturedNews and NewsCard still look clean.

---

2. API Returns No Articles

Case: NewsAPI might respond with an empty array due to:

- Rate limit exceeded
- Server issues
- Region filters returning no news

Handling:

```
if (topNews.length === 0) {
  <p>No news available.</p>
}
```

Result:

- Homepage and category pages show a simple safe fallback message.
- The UI never crashes.

---

3. Long Titles Breaking Layout

Case: Certain news headlines are extremely long.

Handling:

Tailwind utilities:

```
line-clamp-2
text-ellipsis
overflow-hidden
```

Result:

- Titles never overflow outside cards.
- Sidebar scroll and grid layout remain aligned.

---

4. When NewsAPI Fails Completely

Case:

- API key invalid
- Network error
- NewsAPI downtime

Handling:

In `getServerSideProps()`:

```
try {
  const articles = await fetchNewsAPI();
} catch (error) {
  return { props: { topNews: [], trendingTopics: [] } };
}
```

Result:

- Website still renders.
- Shows proper fallback messages (No news available).
- Prevents hydration errors.

---

6. Category Without Matching News

Case:

User visits a category page where no article matches the category.

Handling:

```
{filteredNews.length === 0 && (
  <p>No news available in this category.</p>
)}
```

Result:

- No crashes
- Clean UI message

---

7. Scrolling Sidebar Edge Case

Ensured scroll only appears on large screens:

```
lg:overflow-y-scroll
overflow-visible
lg:h-[720px]
```

Result:

- Mobile layouts remain natural
- Desktop sidebar scrolls perfectly

PART D – AI Use + Reflection

This project was completed primarily through my own coding, debugging, and design work. Only around **15% of the work involved assistance from AI tools**, mainly for support tasks, not core implementation.

Below is an honest and detailed reflection on how AI was used and how I ensured correctness.

---

1. Parts of the Assignment Where AI Was Used (≈15%)

a) Documentation Writing

AI helped in:

- Structuring Part B documentation
- Rewriting wireframe descriptions
- Organizing layout and design explanation
- Writing this Part D section

AI was used only to **format and refine text**, not to write code.

---

b) Folder Structure Planning

AI suggested:

- A clean folder layout
- Separation of components (`layout`, `common`, `news`)
- Placement of mock data and types

I used these suggestions but **implemented the full structure myself**.

---

c) Generating Mock Data

AI helped generate:

- A consistent mock dataset
- Multiple categories with sample news items

But I:

- Verified all values
- Fixed broken image URLs
- Adjusted categories and titles manually

---

d) Debugging Assistance

AI helped identify and explain:

- Next.js image domain configuration error
- Serialization issues with `undefined` fields
- Folder mismatch and mockData loading problems

However, the debugging and fixes were implemented manually.

---

2. Where AI Suggestions Were Wrong or Suboptimal

AI provided some suggestions that did **not work correctly**, such as:

a) Incorrect Next.js Image Config

AI initially suggested using:

```
domains: ["*"]
```

(which is invalid)

I researched and replaced it with:

```
remotePatterns: [{ protocol: "https", hostname: "**" }]
```

b) API Normalization Logic

AI's mapping logic produced:

- `undefined` values
- Missing fallbacks
- Incorrect slugs

I manually rewrote the mapping function and finally removed API usage entirely.

c) Dynamic Routing Issues

AI suggested slug-based paths using NewsAPI, which is impossible because NewsAPI gives no slugs.

I corrected this by:

- Using mock IDs
- Creating clean dynamic pages with predictable paths

---

3. How I Verified and Corrected AI Suggestions

✓ Manual Testing

I manually tested:

- UI layout
- Responsive design
- MockData consistency
- All pages with and without data

✓ Rewriting Code Myself

AI sometimes produced generic or non-optimized code.
I improved it by:

- Simplifying components
- Adding TypeScript types
- Cleaning unused imports
- Adjusting layout logic (especially mobile view)

✓ Cross-checking Documentation

AI-generated explanations were corrected by:

- Removing inaccuracies
- Adjusting to match my actual project
- Rewriting final content in my own words

---

4. Custom Modifications I Made Beyond AI Suggestions

a) Full UI Implementation

I built:

- FeaturedNews
- NewsCard
- NewsGrid
- Navigation
- TrendingTopics
  from scratch using TailwindCSS and Next.js.

b) Responsive Layout

Mobile and desktop breakpoints were completely hand-coded by me.

c) Clean TypeScript Models

I created:

- Types for mock news
- Category mapping
- Correct interfaces

d) Better Folder Architecture

I reorganized:

- Components
- Pages
- Data
- Types

in a much cleaner structure than what AI proposed.

e) Switching from API to MockData

I made the strategic decision to drop NewsAPI and maintain local mockData for stability.

Final Reflection

AI helped **only for supporting tasks**, not core development.
Around **85% of the work—including coding, debugging, layout building, UI design, responsiveness, logic, and routing—was done manually by me.**

AI mainly served as:

- A helper for documentation
- A debugging assistant
- A generator for sample mock data
- A reference for structuring ideas

All final code and implementation were independently written, tested, and refined by me.